Assignment 2 Report

AI at Scale Lab - ID5003W

—--

SOUMYA MUKHERJEE | CH24M571

—--

This report details the findings and result of a series of operations on banking data using spark ml library . The analysis involved significant data cleaning, preprocessing, model training, and performance profiling.

In the data given in the server, **Logistic Regression** emerged as the most accurate predictor with **83.7% accuracy**. A separate performance profiling analysis was conducted to assess the impact of Spark configurations on runtime. Interestingly, for the profiling workload, increasing executor cores and the degree of parallelism resulted in a slight *increase* in the overall processing time, suggesting that the overhead of distributing tasks outweighed the benefits of parallel execution for that specific job.

1. Model Development & Performance

Data Processing and Dataset Overview

The initial dataset underwent a significant cleaning process before analysis. While running the code , errors observed in the logs indicated presence of null values . The following were observed during the analysis.

- **Initial Row Count:** 3,165,211
- **Data Cleaning:** 3,120,000 rows with null 'y' values were removed.
- **Final Row Count:** 45,211
- **Subscription Counts:**
  - No: 39,922
  - Yes: 5,289
- **Train/Test Split:**
  - Training set size: 36,184

    - Test set size: 9,027
- **Class Imbalance:** The dataset shows a notable class imbalance, with a **7.62-to-1 ratio** of non-subscribers to subscribers. This was addressed during the modeling phase by using class weights as a means to address the imbalance . Initially , I started off with SMOTEENC but since the spark cluster doesn't support installing external libraries (imbalance-learn in this case) , I had to drop the idea . Based on the given sample data , SMOTEENC performed better than using class weights .

Model Performance

Two classification models were trained and evaluated. **Logistic Regression** was identified as the best-performing model among Random forest.

- **Best Model: Logistic Regression**
- **Logistic Regression Metrics:**
    - **Accuracy:** 0.8366
    - **Precision:** 0.9028
    - **Recall:** 0.8366
- **Random Forest Metrics:**
    - **Accuracy:** 0.8169
    - **Precision:** 0.8968
    - **Recall:** 0.8169
    -

Random Forest (with hyperparameter tuning):

Accuracy : 0.8287

Thus accuracy increased slightly with hyperparameter tuning .

Due to the long run-time of the fine tuning portion , number of trees were reduced to complete the execution .

Confusion Matrix (Logistic Regression):

The confusion matrix for the best model shows its performance in correctly and incorrectly classifying subscriptions.

[[6663, 1272],

 [203, 889]]

Hyperparameter Selection for Random Forest:

For Random Forest, I selected the following hyperparameters for tuning:

- **numTrees**: This is the number of decision trees in the forest. More trees generally lead to better performance but increase computation time. I chose values [10, 20] to explore the impact of increasing the number of trees.
- **maxDepth**: This parameter controls the maximum depth of each tree. Deeper trees can capture more complex relationships but are prone to overfitting. I selected [5, 10] to investigate different levels of tree complexity and its effect on the model's performance and potential overfitting.
- **minInstancesPerNod**e: This determines the minimum number of instances a node must have to be split. A higher value prevents the creation of nodes with very few instances, which can help reduce overfitting. I used [5, 10] to see how this affects the tree structure and model generalization.

These hyperparameters were chosen because they are among the most influential parameters for Random Forest performance and directly impact the trade-off between model complexity and generalization.

2. Performance Profiling: The Impact of Parallelism

The effect of different Spark configurations on runtime was tested in a separate analysis. The key takeaway from this run is that for that particular workload, **increasing parallelism did not reduce the total execution time**.

Configuration	**Data Loading (s)**	**Preprocessing (s)**	**Model Training (s)**	**Model Evaluation (s)**	**Accuracy**	**Total Time (s)**
Cores: 1, Max: 2, Mem: 1g	1.409	117.547	53.842	11.517	0.890551	184.315
Cores: 2, Max: 2, Mem: 1g	1.778	124.525	54.056	11.423	0.890551	191.783

Cores: 2, Max: 2, Mem: 1g, Parallelism: 1	1.021	125.993	53.999	11.414	0.890551	192.428
Cores: 2, Max: 2, Mem: 1g, Parallelism: 2	0.902	125.436	55.575	11.928	0.890551	193.841

Analysis of Runtimes:

- **Increasing Cores:** Simply increasing executor cores from 1 to 2 (Configuration 1 vs. 2) did not improve performance; instead, the total time increased by approximately 4%.

- **Introducing Parallelism:** Adding and increasing the degree of parallelism (Configurations 3 and 4) also resulted in slightly longer total runtimes compared to the single-core configuration.

This unexpected result suggests that for the size of that dataset and the complexity of the operations, the overhead associated with managing and distributing tasks was greater than the computational gains from parallel execution.

Conclusion

The analysis successfully identified **Logistic Regression** as the most effective model for predicting customer subscriptions based on the provided data. The separate performance profiling revealed an important insight: more resources do not always guarantee faster results. In that specific test, the overhead of distributed processing in Spark slightly outweighed the benefits, leading to longer runtimes with increased parallelism. This highlights that performance tuning is not just about adding more resources, but about finding the optimal configuration for a specific workload.

Log Files :

- CH24M571-Config1.log - executor_cores": "1", "max_cores": "2", "executor_memory": "1g", "parallelism": None
- CH24M571-Config2.log - executor_cores": "2", "max_cores": "2", "executor_memory": "1g", "parallelism": None

- CH24M571-Config3.log- ‘executor_cores": "2", "max_cores": "2", "executor_memory": "1g", "parallelism": "1"

- CH24M571-Config4.log - ‘executor_cores": "2", "max_cores": "2", "executor_memory": "1g", "parallelism": "2"
- CH24M571-MainFile.log - Log containing the main part of the code Q1 to 4